ECE421: Introduction to Machine Learning — Fall 2024

Assignment 3: Feed-forward Neural Network

Due Date: Friday, November 15, 11:59 PM

General Notes

1. Programming assignments can be done in groups of up to 2 students. Students can be in different sections.
2. Only one submission from a group member is required.
3. Group members will receive the same grade.
4. Please post assignment-related questions on Piazza.

Group Members

Name (and Name on Quercus)	UTORid
Shulin Ji	jishuli1
Ruoheng Wang	wangr279

1 Implementing Feed-Forward Neural Network Using `NumPy`

1.1 Basic Network Layers

1.1.1 ReLU

1.1.1.a Derive the gradient of the downstream loss with respect to the input of the ReLU activation function, Z. You must arrive at a solution in terms of $\partial L/\partial Y$, the gradient of the loss w.r.t. the output of ReLU $Y = \sigma_{\text{ReLU}}(Z)$, and the batched input Z, *i.e.*, where $Z \in \mathbb{R}^{m \times n}$. Include your derivation in your writeup. [**HINT:** you are allowed to use operations like elementwise multiplication and/or division!]

Answer.

1. Derivative of ReLU

The derivative of $\sigma_{\text{ReLU}}(z)$ with respect to z is:

$$\frac{\partial Y}{\partial Z} = \begin{cases} 1 & \text{if } Z > 0 \\ 0 & \text{if } Z \leq 0 \end{cases}$$

Express this elementwise derivative using an indicator function:

$$\frac{\partial Y}{\partial Z} = \mathbf{1}_{Z>0}$$

where $\mathbf{1}_{Z>0}$ is a matrix of the same dimensions as Z, containing 1 for elements where $Z > 0$ and 0 otherwise.

2. Chain Rule for the Gradient

Find $\frac{\partial L}{\partial Z}$, the gradient of the loss L with respect to the input Z. By chain rule, we have:

$$\frac{\partial L}{\partial Z} = \frac{\partial L}{\partial Y} \odot \frac{\partial Y}{\partial Z}$$

3. Substitute the ReLU Derivative

Substitute $\frac{\partial Y}{\partial Z} = \mathbf{1}_{Z>0}$:

$$\frac{\partial L}{\partial Z} = \frac{\partial L}{\partial Y} \odot \mathbf{1}_{Z>0}$$

1.1.2 Fully-Connected Layer

1.1.2.a Derive the gradients of the loss $L \in \mathbb{R}$ with respect to weight matrix $W \in \mathbb{R}^{n^{[l]} \times n^{[l+1]}}$, i.e., $\partial L / \partial W$, and with respect to the bias row vector $\mathbf{b} \in \mathbb{R}^{1 \times n^{[l+1]}}$, i.e., $\partial L / \partial \mathbf{b}$, in the fully-connected layer. You will also need to take the gradient of the loss with respect to the input of the layer $\partial L / \partial X$, which will be passed to lower layers, where $X \in \mathbb{R}^{m \times n^{[l]}}$ is the batched input. Again, you must arrive at a solution that uses batched X and Z. Please express your solution in terms of $\partial L / \partial Z$, which you have already obtained in question 1.1.1.a, where $Z = XW + \mathbf{1}^\top \mathbf{b}$ and $\mathbf{1} \in \mathbb{R}^{1 \times m}$ is a row of ones. Note that $\mathbf{1}^\top \mathbf{b}$ is a matrix whose each row is the row vector $\mathbf{b}$. So, we are adding the same bias vector to each sample during the forward pass: this is the mathematical equivalent of numpy broadcasting. Include your derivations in your writeup.

Answer.
1. Gradient with respect to W
By chain rule, the derivative of Z with respect to W is:

$$\frac{\partial Z}{\partial W} = X$$

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial Z} \cdot \frac{\partial Z}{\partial W} = X^\top \frac{\partial L}{\partial Z}$$

2. Gradient with respect to b
b is broadcasted across each row of Z in the forward pass. We can sum the gradient of L with respect to Z across all rows to get the gradient with respect to b:

$$\frac{\partial L}{\partial b} = \sum_{i=1}^{m} \frac{\partial L}{\partial Z_{i,:}}$$

in matrix notation:

$$\frac{\partial L}{\partial b} = \mathbf{1} \frac{\partial L}{\partial Z}$$

3. Gradient with respect to X
Differentiate $Z = XW + \mathbf{1}^\top b$ with respect to X:

$$\frac{\partial L}{\partial X} = \frac{\partial L}{\partial Z} W^\top$$

1.1.3 Softmax Activation

1.1.3.a For a single training point, derive $\partial\sigma_i/\partial s_j$ for an arbitrary (i, j) pair, *i.e.* derive the Jacobian Matrix

$$\begin{bmatrix} \frac{\partial\sigma_1}{\partial s_1}, & \frac{\partial\sigma_1}{\partial s_2}, & \ldots, & \frac{\partial\sigma_1}{\partial s_k} \\ \frac{\partial\sigma_2}{\partial s_1}, & \frac{\partial\sigma_2}{\partial s_2}, & \ldots, & \frac{\partial\sigma_2}{\partial s_k} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial\sigma_k}{\partial s_1}, & \frac{\partial\sigma_k}{\partial s_2}, & \ldots, & \frac{\partial\sigma_k}{\partial s_k} \end{bmatrix}.$$

Include your derivation in your writeup. You do not need to use batched inputs for this question; an answer for a single training point is acceptable.

Answer.
We consider two cases $i = j$ and $i \neq j$.

Case 1: $i = j$

$$\frac{\partial\sigma_i}{\partial s_i} = \frac{\partial}{\partial s_i}\left(\frac{e^{s_i}}{\sum_{j=1}^{k} e^{s_j}}\right)$$

$$\frac{\partial\sigma_i}{\partial s_i} = \frac{\left(e^{s_i} \cdot \sum_{j=1}^{k} e^{s_j}\right) - \left(e^{s_i} \cdot e^{s_i}\right)}{\left(\sum_{j=1}^{k} e^{s_j}\right)^2}$$

$$\frac{\partial\sigma_i}{\partial s_i} = \frac{e^{s_i}\left(\sum_{j=1}^{k} e^{s_j} - e^{s_i}\right)}{\left(\sum_{j=1}^{k} e^{s_j}\right)^2}$$

$$\frac{\partial\sigma_i}{\partial s_i} = \sigma_i\left(1 - \sigma_i\right)$$

Case 2: $i \neq j$

$$\frac{\partial\sigma_i}{\partial s_j} = \frac{\partial}{\partial s_j}\left(\frac{e^{s_i}}{\sum_{k=1}^{n} e^{s_k}}\right)$$

$$\frac{\partial\sigma_i}{\partial s_j} = \frac{0 \cdot \sum_{k=1}^{n} e^{s_k} - e^{s_i} \cdot e^{s_j}}{\left(\sum_{k=1}^{n} e^{s_k}\right)^2}$$

$$\frac{\partial\sigma_i}{\partial s_j} = -\sigma_i\sigma_j$$

$$J_{ij} = \begin{cases} \sigma_i\left(1 - \sigma_i\right) & \text{if } i = j \\ -\sigma_i\sigma_j & \text{if } i \neq j \end{cases}$$

1.1.4 Cross-Entropy Loss

1.1.4.a Derive $\partial L/\partial \hat{Y}$ the gradient of the cross-entropy cost with respect to the network's predictions, $\hat{Y}$. You must use batched inputs. Include your derivation in your writeup.
[**HINT:** You are allowed to use operations like elementwise multiplication and/or division!]

Answer.
1. Derive the gradient $\frac{\partial L}{\partial \hat{Y}}$

$$L = -\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{k} Y_{ij} \ln \hat{Y}_{ij}$$

$$\frac{\partial L}{\partial \hat{Y}_{ij}} = -\frac{1}{m} \cdot \frac{Y_{ij}}{\hat{Y}_{ij}}$$

2. Express gradient in matrix form

$$\frac{\partial L}{\partial \hat{Y}} = -\frac{1}{m} \cdot \frac{Y}{\hat{Y}}$$

where the division $\frac{Y}{\hat{Y}}$ is elementwise

1.2 Two-Layer Fully Connected Networks

1.2.1 Implementing Fully Connected Layer

1.2.2 Hyperparameter Tuning (in a very small scale)

1.2.2.a try at least 3 different combinations of these hyperparameters. Report the results of your exploration, including the values of the parameters you explored and which set of parameters gave the best test error. Provide plots showing the loss versus iterations for your best model and report your final test error.

Answer.

- The results of the exploration (including the values of the parameters we explored):

Combination	Learning Rate	Momentum	Activation Function	Hidden Layer Size
1	0.01	0.9	Relu	5
2	0.005	0.9	Relu	10
3	0.005	0.8	Sigmoid	10
4	0.005	0.85	Relu	20

Table 1: Hyperparameter Combinations

Combination	Test Loss	Test Accuracy
1	7.1849	0.74
2	0.1116	0.96
3	0.557	0.9
4	4.974	0.82

Table 2: Test Results

- The figure below shows the loss versus iterations for my best model.
- The set of parameter that gave the best test error is: Learning Rate: 0.005 Momentum: 0.9, Activation Function: Relu, Hidden Layer Size: 10

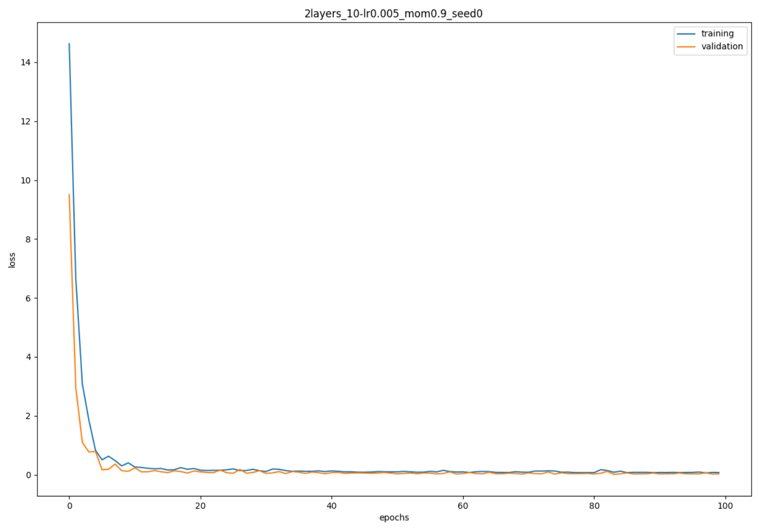

Figure 1: Loss versus iterations for the best model.

2 Implementing Neural Networks Using `PyTorch`

2.1 Understanding `PyTorch`

2.2 Implementing a Multi-Layer Perceptron Model Using `PyTorch`

2.2.a Code for training an MLP on MNIST (you should upload your ipynb notebook and also provide your code for this section in `PA3_qa.pdf`, as a screen shot, or in latex typesetting, etc.).

Answer.

```
images = [training_data[i][0] for i in range(9)]
plt.imshow(
            torchvision.utils.make_grid(torch.stack(images),
            nrow=3,
            padding=5
            ).numpy().transpose((1, 2, 0)))
### YOUR CODE HERE ###

# Imports for pytorch
import numpy as np
import torch
import torchvision
from torch import nn
import matplotlib
from matplotlib import pyplot as plt
from tqdm.notebook import tqdm
import torch.nn.functional as F


# Creating the datasets
transform = torchvision.transforms.ToTensor()
```

```
training_data = torchvision.datasets.FashionMNIST(
    root="data",
    train=True,
    download=True,
    transform=transform,
)

validation_data = torchvision.datasets.FashionMNIST(
    root="data",
    train=False,
    download=True,
    transform=transform,
)

# Set up device to use GPU if possible
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using device", device)

# Set up training and validation dataloaders for batch training
batch_size = 64
train_loader = torch.utils.data.DataLoader(training_data, batch_size=
   batch_size, shuffle=True)
val_loader = torch.utils.data.DataLoader(validation_data, batch_size=
   batch_size, shuffle=False)

# Define our 3-layer artificial neural network (MLP) model
class FashionMLP(nn.Module):
    def __init__(self):
        super(FashionMLP, self).__init__()
        # Input: 784 (28x28), Output: 128 neurons
        self.layer1 = nn.Linear(28 * 28, 128)
        # Hidden layer: 128 -> 64 neurons
        self.layer2 = nn.Linear(128, 64)
        # Output layer: 64 -> 10 (Use 10 as final output layer since there
             are 10 classes in FashionMNIST)
        self.layer3 = nn.Linear(64, 10)


    def forward(self, img):
        # Flatten the image into a vector
        flattened = img.view(-1, 28 * 28)
        # First hidden layer, apply Relu activation function
        activation1 = F.relu(self.layer1(flattened))
        # Second hidden layer, apply Relu activation function
        activation2 = F.relu(self.layer2(activation1))
        # Output layer (no activation function here)
        output = self.layer3(activation2)
        return output


# Instantiate the model and move it to the appropriate device
model = FashionMLP().to(device)

# Optimizer (SGD) and loss function (cross entropy)
```

```
learning_rate = 0.03
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
loss_func = nn.CrossEntropyLoss()

epochs = 20  # Increased epochs to try to achieve 82% accuracy
train_losses = [] # store training losses for plotting
val_losses = [] # store validation losses for plotting
train_accuracies = [] # store training accuracies for plotting
val_accuracies = [] # store validation accuracies for plotting

# Training loop
model.train()  # Put model in training mode
for epoch in range(epochs):
    training_losses = []
    correct_train = 0
    total_train = 0
    for x, y in tqdm(train_loader, unit="batch"):
        x, y = x.to(device), y.to(device)  # Move data to the device
        optimizer.zero_grad()  # Clear previous gradients
        pred = model(x)  # Forward pass
        loss = loss_func(pred, y)  # Compute loss by nn.CrossEntropyLoss
        loss.backward()  # Backpropagate to compute gradients
        optimizer.step()  # Update model parameters

        # Store training loss
        training_losses.append(loss.item())

        # Compute training accuracy
        _, predicted = torch.max(pred.data, 1)
        total_train += y.size(0)
        correct_train += (predicted == y).sum().item()

    train_losses.append(np.mean(training_losses))
    train_accuracies.append(correct_train / total_train)

    # Compute the validation loss with nn.CrossEntropyLoss()
    model.eval()  # Set model to evaluation mode
    validation_losses = [] # store the validation losses for plot
    correct_val = 0
    total_val = 0
    with torch.no_grad():
        for x_val, y_val in val_loader:
            x_val, y_val = x_val.to(device), y_val.to(device)
            pred_val = model(x_val)  # Forward pass for validation data
            val_loss = loss_func(pred_val, y_val) # Apply cross entropy
                loss function to it
            validation_losses.append(val_loss.item())

            # Compute validation accuracy
            _, predicted_val = torch.max(pred_val.data, 1)
            total_val += y_val.size(0)
            correct_val += (predicted_val == y_val).sum().item()

    val_losses.append(np.mean(validation_losses))
```

```
        val_accuracies.append(correct_val / total_val)

        print(f'Epoch {epoch+1}, '
              f'Train Loss: {train_losses[-1]:.4f}, Train Acc: {
                  train_accuracies[-1]:.4f}, '
              f'Val Loss: {val_losses[-1]:.4f}, Val Acc: {val_accuracies
                  [-1]:.4f}')

        model.train()  # Set model back to training mode after validation


    # Plotting the results

    # Plot training and validation loss for the first 10 epochs
    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
    plt.plot(range(1, 11), train_losses[:10], label="Training Loss")
    plt.plot(range(1, 11), val_losses[:10], label="Validation Loss")
    plt.title("Loss over First 10 Epochs")
    plt.xlabel("Epochs")
    plt.ylabel("Loss")
    plt.legend()
    plt.show()


    # Plot training and validation accuracy over all epochs
    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 2)
    plt.plot(range(1, epochs+1), train_accuracies, label="Training Accuracy")
    plt.plot(range(1, epochs+1), val_accuracies, label="Validation Accuracy")
    plt.title("Accuracy over Epochs")
    plt.xlabel("Epochs")
    plt.ylabel("Accuracy")
    plt.legend()
    plt.show()

    # Return final accuracy for validation
    final_accuracy = val_accuracies[-1]
    print("Final Validation Accuracy is {}%".format(final_accuracy * 100))
```

2.2.b A plot of the training loss and validation loss for each epoch of training after trainnig for at least 8 epochs.

Answer.

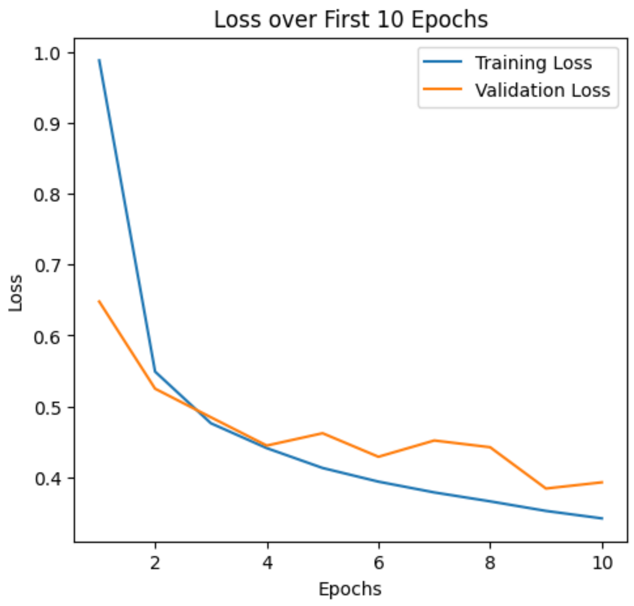

Figure 2: Training loss and validation loss for each epoch.

2.2.c A plot of the training and validation accuracy, showing that it is at least 82% for validation by the end of training.

Answer.

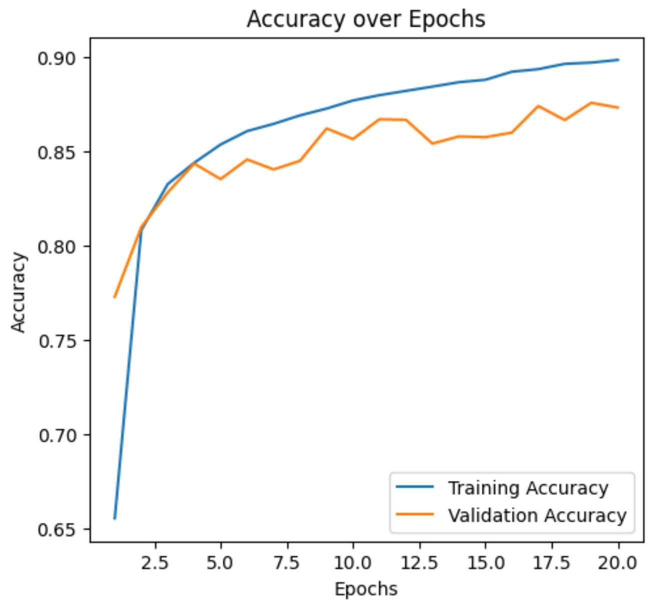

Figure 3: Training and validation accuracy, showing that it is at least 82% for validation by the end of training.

3 Discussion

Please answer the following short questions so we can improve future assignments.

3.a How much time did you spend on each part of this assignment?

Answer. Our team members both spend around 7 - 8 hours on the assignment.

3.b Any additional feedback? How would you like to modify this assignment to improve it?

Answer. We hope there could be more guides on how to construct our neural networks to improve accuracy.